THE BELL SYSTEM TECHNICAL JOURNAL
SAFEGUARD SUPPLEMENT
*Printed in U.S.A.*

## *SAFEGUARD Data-Processing System:*

# The Dictionary Approach to Digital Maintenance

By C. J. RIFENBERG

(Manuscript received January 3, 1975)

*This paper provides an overview of one aspect of the SAFEGUARD approach to digital maintenance—the Maintenance and Diagnostic (M&D) program-fault dictionary. The M&D program detects the presence of faults. The associated fault dictionary provides fault lists for automatic fault isolation; it is generated by executing the maintenance program in an environment simulating the action of hardware in the presence of faults. The paper also provides some detailed discussion of simulator-performance improvements.*

### I. INTRODUCTION

A SAFEGUARD data-processing system consists of racks of equipment for three functional areas: a large real-time central computer facility, a large peripheral subsystem, and a Maintenance and Diagnostic Subsystem (M&DSS).[1,2] This paper describes an essential aspect of the SAFEGUARD maintenance plan, the Maintenance and Diagnostic program-fault dictionary.*

Fault-isolation dictionaries are available for most maintenance programs. Dictionaries provide a correspondence between fault-diagnostic-test failures and possible hardware faults (or faults of replaceable units) which could cause the failures. They have been used successfully in the No. 1 Electronic Switching System (ESS)[3,4]; however, ESS and SAFEGUARD dictionaries differ in their format, generation, and use. Both Armstrong[5] and Godoy[6] have described a method for efficiently simulating the action of hardware in the presence of faults. Their technique is used in the generation of SAFEGUARD fault-isolation dictionaries.

* Maintenance and Diagnostic programs are described by Hahn and Slojkowski.[1] In addition, supplemental maintenance programs are used to test hardware, which cannot be exercised by these programs, or to provide increased fault detection.

A test-control program accesses the dictionaries to isolate detected faults. After receiving results of test failure, the test-control program performs set union and intersection operations on the sets of fault lists in the dictionary entries associated with failed and passed tests to isolate them to an acceptable number of replaceable units (chassis).* If a maintenance program is completed without failure, the test-control program can either consider the rack fault free, schedule additional maintenance programs for execution within the M&D controller, or schedule supplemental maintenance programs.

## II. CONSTRUCTION OF SAFEGUARD DICTIONARIES

### *2.1 Approach*

The dictionary approach to fault isolation was chosen early in the design cycle primarily to satisfy a requirement that craftspeople with moderate skill, working at a large number of installations, be able to quickly accomplish fault isolation. Simulation was considered as the only feasible method for generating dictionaries since there was no hardware time available for fault insertion, and the logic was too complex for manual dictionary generation.

Figure 1 is a block diagram of the Logic Simulation Facility (LSF). For simulation purposes, each rack is divided into several, often overlapping, logic blocks, none of which exceeds 20,000 gates. This maximum gate count is a serious design limitation which occasionally causes functionally integral logic blocks to be subdivided. Had time permitted, this design limitation would have been eliminated. Each SAFEGUARD data-processing system has over 300 maintenance programs designed to detect faults in the logic blocks. The tests within the program are designed manually. Most of these programs have associated fault-isolation dictionaries generated through simulation. A few programs (mostly for rack interface blocks) were not simulated since they were only testing a small portion of a block functionally much larger than 20,000 gates. Before programs are run on the simulator, they are debugged on the hardware to verify that predetermined logic values within compare instructions are correct. By debugging on the hardware rather than the simulator, the possibility that the simulated logic block is incorrectly constructed or initialized is eliminated.

Circuit interconnections and other pertinent wiring information for the computer units are described in manufacturing tape files. The data in these files are used by an automatic wire-wrap machine to wire the

* These chassis (500 to 600 logic gates) are, in turn, repaired by replacing integrated-circuit packages.

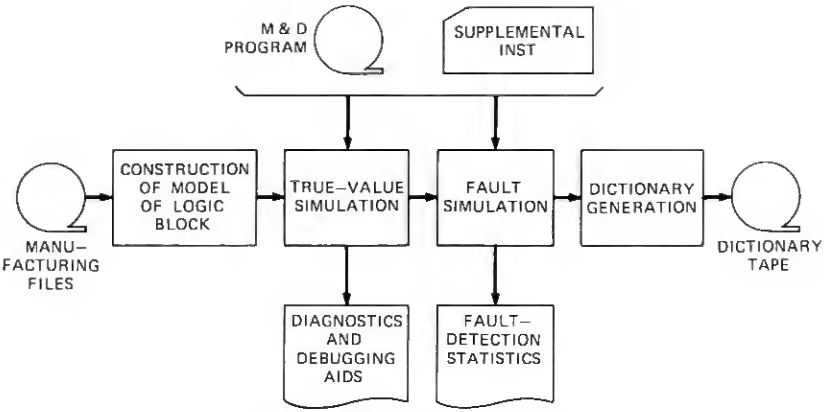


Fig. 1—Logic simulation facility.

chassis and racks. These files are used to construct a simulation data base and to simulate the hardware at the logic-gate level.

In addition to the manufacturing files, there are two primary inputs to simulation: the maintenance program discussed above and a set of supplemental instructions. These supplemental instructions enable the test designer to set any gate in the simulated logic to any logic state. They are particularly useful in initializing gates on the boundary of the logic block which are driven from logic not being simulated. It is through these instructions that the simulated logic block goes from an unknown state to a state representing the hardware at the start of testing.

The true logic value simulation, pictured in Fig. 1, is a simulated execution of the maintenance program in the absence of faults. Since the program has "run clean" on the hardware (i.e., all compare instructions are correct in predicted true logic value), the true logic value simulation is used to find discrepancies between simulation and hardware execution of the maintenance program. Discrepancies are usually caused by erroneous supplemental instructions or by deficiencies in the logic-block data base. These differences are usually resolved through changes to the instructions or data base. Standard aids are provided to assist in identifying causes for discrepancies (e.g., gate timing traces of change from known to unknown logic value).

The LSF fault simulator is a deductive simulator (see Ref. 3). At any given simulation time, each gate in the circuit has a true logic value (possibly unknown) and a fault list (possibly null) associated with it. A gate's fault list contains all faults in the circuit which, if present singly, would complement the true logic value of the gate. Every fault present in a gate's fault list is said to be detectable at the

gate. The simulator assumes that only single, hard faults occur in the hardware. Transient failures, most timing faults, and marginal faults are not considered. Unit gate delay is assumed. At each interval of simulation time, the fault-free logic value and the fault list for a gate are computed if either the logic value or fault list of any of the gate's inputs has changed in the preceding time period. When a compare instruction of the maintenance program is simulated, the instruction number and all faults associated with the compared register (i.e., all the faults which, if present singly, would cause a bit to be complemented from its true logic value) are output to a fault tape for later dictionary generation. Thus, for each compare instruction, there exists a list of faults which are detected by that compare instruction due to their causing an incorrect logic value in the compared register.

Statistical programs provide the maintenance programmer with both summary and detailed information on the faults detected and faults simulated but not detected. This output from the simulator, in many cases, is more important than the dictionary (described in Section III) and is a significant advantage of the simulation approach to dictionary generation. The statistical information is used locally to improve the detection quality of a given program. It is used globally in directing efforts to improve detection in certain areas (e.g., to design a supplemental maintenance program) or conversely, to suspend effort in an area already achieving good detection.

### 2.2 Simulation performance improvements

The initial version of the simulation facility required extensive computer usage for dictionary generation. Estimates indicated full utilization of an HIS 635 computer for a period of about two years. Even this large cost was an underestimate since many programs would have to be simulated more than once either because the corresponding hardware was significantly changed or because the program was significantly modified to improve detection. Therefore, considerable effort was devoted to reducing computing requirements. Some resource-use reduction resulted from internal algorithm and code modification. The four major items below, however, have most significantly reduced resource requirements, with a cumulative effect of approximately a ten-fold reduction.

#### 2.2.1 Fault list paging

Core storage requirements for fault lists can become excessive. This necessitates partitioning of the simulation into $n$ fault runs, each simulating faults in only $1/n$ of the total number of gates. Results for partitions are merged into a single dictionary. Since the entire M&D

Table I — Computer time savings due to software paging

| Block | Partitions | | Total Elapsed Hours | |
|---|---|---|---|---|
| | No Paging | Paging | No Paging | Paging |
| 1 | 100 | 13 | 210.0 | 110.0 |
| 2 | 50 | 12 | 19.4 | 10.0 |
| 3 | 20 | 6 | 11.1 | 3.5 |
| 4 | 6 | 1 | 7.3 | 2.2 |
| 5 | 4 | 1 | 5.1 | 1.9 |
| 6 | 4 | 1 | 4.0 | 1.3 |

program must be simulated for each partition, the time required for calculating the true logic values is multiplied by the number of partitions. When $n$ becomes large this introduces a very significant overhead. However, it was determined by experimentation that core requirements for fault lists during simulation peak sharply after a few tests and then fall off quickly (particularly after implementation of other performance improvements to be described). An objective of reducing the number of partitions and total elapsed time was then met by a fault-list paging algorithm which minimized the time required during the absence of paging at the expense of time required during demand paging. The number of partitions for very large blocks is not always reduced to one in order to prevent the paging overhead from exceeding the overhead inherent in dividing the block into a few partitions. On the average, the number of partitions required is reduced by about 75 percent while elapsed computer time is reduced by 40 to 75 percent. Table I provides some sample computer time savings due to demand paging of fault lists.

#### 2.2.2 No simulation of conditionals

In simulation, unknown logic values appearing on the output of gates can be due to either one or more uninitialized boundary-access terminals or to a race condition in a flip-flop. The fault list associated with a node whose state is unknown is not unique, since detection of a fault is dependent upon the particular logic value present. Armstrong[5] provides a method for nonexact treatment of fault lists in the presence of unknowns in order to reduce simulation time. The method was successful because the majority of unknowns appear only transiently and are replaced by known states before monitoring is performed. This method flags faults as "conditional" if their detection is conditioned on the logic value actually existing at an unknown input. It provides a more accurate simulator than one which ignores conditionals.

Experimentation was performed on the trade-off involved between computer time required for simulation of conditionals versus decrease

in fault isolation by nonsimulation of conditionals. Simulation of conditionals required from four to ten times as much computer time as did nonsimulation of conditionals. An additional 3 to 5 percent of the faults in the test blocks had no chassis isolation or wrong chassis isolation when dictionaries were generated without simulating conditional faults. It was concluded that conditionals should not be simulated so that computer time could be more profitably used.

#### 2.2.3 Fault elimination

Fault isolation is essentially a process of applying tests and observing passes and failures (i.e., a fault signature) until only faults on an acceptably small number of replaceable units have the same signature. For example, Table II illustrates fault signatures for three faults. Faults a and b are indistinguishable in signature while Fault c is distinguished from a and b at Tests 5 and 9.

The effect upon isolation of not simulating all faults for all tests was investigated; e.g., one could stop simulating a fault after it is detected once or twice (i.e., fails one or two tests). In the example in Table I, if a fault were no longer simulated after one detection, Faults a, b, and c would now be indistinguishable since they have the same signature through the first detection (i.e., Test 4). On the other hand, if the fault were no longer simulated after two detections, Faults a, b, and c would have the same isolation as simulating all faults for all tests, since Fault c is still distinguished from a and b at Test 5.

Several blocks were simulated varying the number of detections required before a fault was eliminated from simulation. Results showed that eliminating a fault after two detections provided dictionaries with essentially the same isolation as eliminating a fault at three or more detections; yet simulation time (all other factors being equal) was reduced by 80 percent compared with no fault elimination. Table III provides some representative statistics. The net simulation time savings is even greater since eliminating faults after two detections

Table II — Sample fault signatures

| Faults | Tests | | | | | | | | | |
|---|---|---|---|---|---|---|---|---|---|---|
| | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 | 10 |
| a | P | P | P | F | F | F | F | P | P | P |
| b | P | P | P | F | F | F | F | P | P | P |
| c | P | P | P | F | P | F | F | P | F | P |

Note: P = Test passes in presence of fault. F = Test fails in presence of fault (detects fault).

Table III — Computer time savings due to fault elimination

| Eliminations* | Savings† | 1‡ | 2‡ | 3‡ |
|---|---|---|---|---|
| 1 | 92 | 83 | 97 | 100 |
| 2 | 88 | 90 | 98 | 100 |
| 3 | 83 | 91 | 98 | 100 |
| No Elim. | — | 92 | 99 | 100 |

* Number of detections prior to elimination.
† Percent simulation time savings vs no elimination.
‡ Percent detected faults isolated to 1, 2, 3 chassis.

contributes to the sharp peaking of core requirements for fault lists and, therefore, is partially responsible for making fault-list paging possible.

#### 2.2.4 Fault collapsing

Another attempt at reducing simulation time was "fault collapsing," i.e., merging two faults if detection of one guarantees detection of the other. For example, consider the string of invertor gates shown in Fig. 2. The effect of the output of C being stuck in logic value one is indistinguishable at the monitorable output from the effect of the output of A being stuck in logic value one. Therefore, a test will either detect both faults or neither fault. If both faults are located on the same replaceable unit, there is no loss in isolation by "collapsing" one onto the other and simulating only one of the faults. In order not to reduce replaceable unit isolation, strong restrictions are placed on candidates for collapsing. Only faults on strings of gates located on a single chassis are considered for fault collapse. Thus, a fault might be isolated to the wrong integrated-circuit package but not the wrong chassis. Typically, 15 percent of the faults in a logic block are collapsed resulting in computer savings of about 10 percent. One problem experienced with this limited fault collapse is that additional time is required to evaluate the accuracy of the simulator, and to repair chassis based on the ambiguous integrated-circuit-package isolation information in the dictionary. Table IV summarizes the results of the parameter trade-offs.

### 2.3 Other methods tried and their limitations

A study of a technique for building dictionaries, called Reachability List Dictionaries (R-LIST), was conducted. Figure 3 is a diagram of

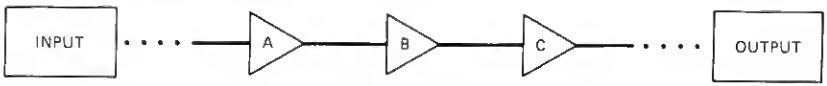


Fig. 2—Sample logic string.

Table IV — Summary of computer time savings vs dictionary degradation

| Parameter | Time* Savings (%) | Dictionary† Degradation (%) |
|---|---|---|
| Paging | 40–75 | 0 |
| No simulation of conditionals | 60–90 | 3–5 |
| Fault elimination | 80–90 | 1 |
| Fault collapse | 10–12 | 0 |

* HIS 635 elapsed computer time vs full simulation.
† Percent of faults with no or incorrect chassis isolation compared with a dictionary created without using the parameter.

a simple logic block. An R-LIST is associated with each output gate (e.g., 4, 5, and 6).

The R-LIST contains all gates (or faults) that lie on paths which feed the gate. The R-LISTS may be derived from the total connectivity matrix for the complete block. The R-LISTS may also be obtained by performing a reverse trace to all input (or boundary) gates to the logic block (e.g., gates 1, 2, and 3 of Fig. 3). The R-LIST can be created from the logic block description alone, without any dynamic simulation. Therefore, there was promise of providing a very economical method of generating dictionaries providing the isolation was good. Experiments were performed to determine the isolation capability of dictionaries constructed using these techniques. They showed poor isolation capability because:

(*i*) Lists were much longer than expected and embraced many chassis. Each list contained over 50 percent of all possible fault-producing gates.
(*ii*) Lists overlapped; that is, many of the gates in any one list appeared in all lists.*

Problems associated with automatically generating tests for large, asynchronous, sequential logic are well known.[7] It is difficult to adapt known test-generation algorithms to such circuits. SAFEGUARD designers were successful, however, in supplementing manually generated tests with automatic addition of compare instructions to outputs not already monitored. Usually, outputs were not monitored because the complexity of the circuit was such that the programmer did not realize the full effect of establishing correct logic configurations on control lines. The simulator was modified to "look" at all output points

* The R-LIST technique was further refined and met with somewhat greater success when applied to ESS 1-A.

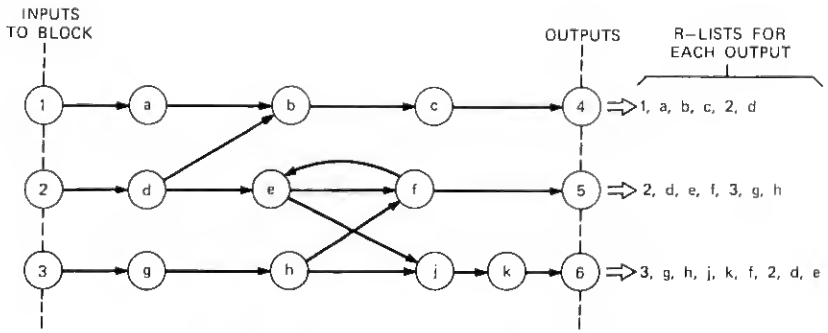


Fig. 3—Sample block with output gate R-LISTS.

for additional propagated faults. This simple technique is being used to increase detection by 3 to 10 percent (an increase which typically required several programmer months).

### III. EXPERIENCE WITH DICTIONARIES

Dictionary entries are associated with test compares which could detect a fault. Functionally, a dictionary entry appears in the form shown in Fig. 4. For example, if Test N failed (i.e., observed output was not $101_2$) with observed error pattern $110_2$, Faults F, G, C, D, and E are candidates for having caused the failure. The test-controller program on the CDC 1700 computer processes dictionary entries corresponding to both matched and mismatched test compares in order to compute a list of faults that have fault signatures consistent with observed test results. The test controller then prints out a list of suspect chassis (with suspect integrated-circuit packages) ordered by chassis with the greatest number of faults on the computed list.

A sample of 31 dictionaries was examined to determine the number of suspect chassis associated with each compare and with each error

TEST COMPARE N
TRUE LOGIC VALUE $101_2$
THREE POSSIBLE ERROR PATTERNS
(1) $000_2$ WITH POSSIBLE FAULTS
A,B CHASSIS 1
C,D,E CHASSIS 2
(2) $110_2$ WITH POSSIBLE FAULTS
F,G CHASSIS 1
C,D,E CHASSIS 2
(3) $111_2$ WITH POSSIBLE FAULTS
A,B CHASSIS 1
H CHASSIS 3

Fig. 4—Functional dictionary entry.

pattern within the compare. For example, in Fig. 4, test compare N shows that faults from three different chassis could cause the test to fail. Figure 4 also shows that faults on only two different chassis could cause any of the three possible error patterns.

These results show that chassis lists are usually short (on the average, 95 percent of the error patterns for a dictionary had three or fewer suspect chassis). Figure 5 indicates that both the tests and the logic are functionally designed; i.e., groups of tests are usually exercising logic that has been reasonably arranged on a small number of chassis. This fact contributes to making the dictionary useful even when there is no exact match between an error pattern in the dictionary and the one occurring during the running of the maintenance program, as is shown below. It also contributes to the success of the above-mentioned performance improvement studies.

Additional testing was performed to determine the accuracy of the simulator and the degree of dictionary isolation. Test approaches included limited hardware fault insertion, comparison with another independent simulator, off-line analysis of dictionaries, and vigorous program testing of simulator versions. The results confirm that the simulator accurately generates dictionaries for hard faults, and dictionaries usually isolate detected hard faults to three chassis more than 90 percent of the time (i.e., if one can detect the hard fault, one can isolate it).

Table V summarizes three different ways of evaluating how well the dictionary approach isolates faults. The first column shows the experimental results from actually inserting 102 randomly chosen faults

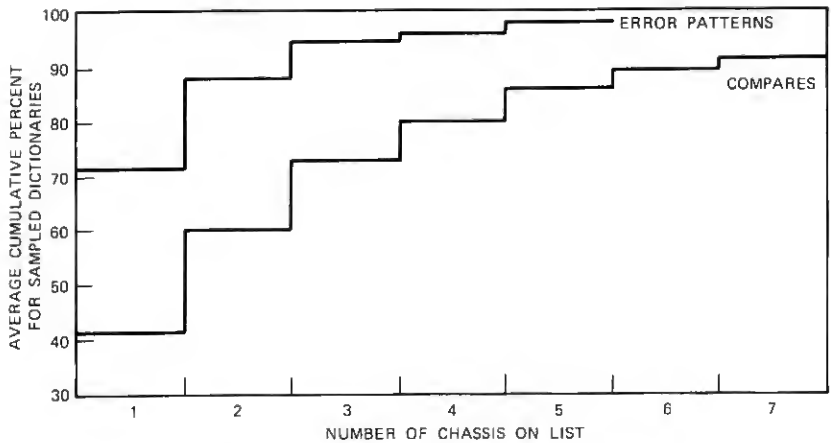


Fig. 5—Typical number of chassis per fault list.

Table V — Preliminary fault isolation evaluation results

| Number of Chassis | Fault Insertion (%) | Independent Simulation (%) | Analysis (%) |
|---|---|---|---|
| 1 | 76 | 70 | 85 |
| 2 or fewer | 87 | 81 | 96 |
| 3 or fewer | 93 | 92 | 98 |
| 4 or fewer | 94 | 94 | 99 |
| 5 or more | 4 | 3 | 1 |
| No or wrong | 2 | 3 | — |

into a processor (99 detected). Physical fault insertion exercised the processor dictionaries in their actual environment. Faults were isolated to three chassis 93 percent of the time. The second column summarizes the results of simulating 261 detected faults on an independent simulator and then searching the appropriate 32 dictionaries for isolation. Finally, the third column summarizes the results obtained by analyzing the 300,000 possible detected faults covered in 19 randomly chosen dictionaries. The size of the 19 logic blocks covered by the dictionaries ranges from 12 to 31 chassis and averages 22 chassis. This analysis assumes that when the M&D program is run on the hardware in the presence of a fault, the first two detections of the fault will occur exactly as predicted in simulation and, therefore, will always yield correct isolation (i.e., the isolation list for a fault is exactly the set of chassis associated with the first two detections). The advantage of this type of analysis is easy determination of the approximate isolation for very large numbers of faults. Again, isolation to three chassis is better than 90 percent.

Dictionary isolation evaluation is continuing with emphasis on increased hardware fault insertion, off-line analysis of dictionaries, and initial field experience. Results to date have been generally consistent with those presented in Table V. In fact, dictionaries have been used in the field to isolate to the integrated-circuit package. The feedback to programmers on detection has been instrumental in improving the quality of program fault coverage. Simulation statistics on processor programs, for example, show they now detect 87 percent of the simulated detectable faults. A four-man committee reviews simulator information on undetected faults and makes recommendations for improvement code. This technique has increased detection by as much as 25 percent in some areas. In most cases, the maintenance program was resimulated after the recommended improvement code was added. In such cases, the simulation data base was first made consistent with the latest hardware changes. In a few cases, where the computer time

for simulation was large, improvement code was added to the end of the program so that the dictionary remained correct with entries corresponding to the added program instructions at the end of the dictionary.

Hardware changes which cause a divergence from the simulated hardware are a significant problem. These hardware changes eventually cause maintenance programs to noncompare when run against fault-free hardware. Such a condition causes rapid modification of the maintenance program. Often, however, the corresponding dictionary cannot be immediately regenerated. Since it is difficult to quantify the resulting dictionary degradation, maintenance personnel eventually lose confidence in the dictionary and stop using it. Dictionaries seem to be worthwhile for hardware that is modified only occasionally.

There has been much discussion about the need for a "nonexact-match"* strategy to handle such items as marginal, transient, and multiple faults or faults improperly handled due to parameter tradeoffs or minor hardware change. The general strategy of on-line processing of dictionary entries allows a very simple algorithm for isolating faults causing exact match. Nonexact match can be handled by interaction between maintenance personnel and dictionary. Simple information requests, such as "List all chassis associated with the first six noncompares or previous six compares," can be answered from the general dictionary entry (see Fig. 4). Such information tells maintenance which logic was being tested at the failed instructions. Since the chassis list is usually short, it is a good starting place for further manual troubleshooting. Thus, maintenance personnel can use the dictionary in homing in on the fault. Not all this interactive capability is currently available. A microfiche print summarizing dictionary entries (i.e., which logic chassis could cause the failure) is being provided to allow such interaction, although less conveniently.

## IV. CONCLUSIONS

As others have noted, simulation facilitates detection feedback. Statistics provided by simulation agree with the laboratory experiments (i.e., they are believable). Since the statistics indicate which faults are not detected, they enable the M&D programmer to improve detection, resulting in a better maintained system. Since good detection is required for good isolation, this benefit of simulation should be considered when one chooses a dictionary generation method. It often

* A nonexact match situation results when a fault causes the maintenance program to noncompare when it is run on the hardware and the dictionary entries do not indicate any fault consistent with the observed error patterns.

overshadows the dictionary itself. If the simulator is efficient, the augmented M&D program can be resimulated.

The consistently high quality of the processor-unit dictionaries, for example, indicates the practicality of dictionaries for large logic blocks (20,000 gates) using SAFEGUARD hardware technology.[1,2] Both the fault model and the simulation were simplified, yet isolation remained quite good. (In fact, multiple faults were often correctly isolated.) Thus, dictionaries for large, stable blocks are useful in isolating faults to a small number of chassis. Because the format actually indicates suspect integrated-circuit packages, the dictionary is further useful in repairing the chassis. On the other hand, dictionaries are marginal, at best, for very small logic blocks, blocks with very low detection, or blocks subject to a very high rate of hardware change activity. Dictionaries can be regenerated for blocks experiencing high hardware change order activity providing the computer time required for regeneration is reasonable.

## REFERENCES


1. J. R. Hahn, Jr., and F. E. Slojkowski, "SAFEGUARD Data-Processing System: Maintenance and Diagnostic Subsystem," B.S.T.J., this issue, pp. S63–S72.
2. J. W. Olson, "SAFEGUARD Data-Processing System: Architecture of the Central Logic and Control," B.S.T.J., this issue, pp. S41–S61.
3. H. Y. Chang, E. Manning, and G. Metze, *Fault Diagnosis of Digital Systems*, New York: Wiley Interscience, 1970.
4. H. Y. Chang and W. Thomis, "Methods of Interpreting Diagnostic Data for Locating Faults in Digital Machines," B.S.T.J., *66*, No. 2 (February 1967), pp. 289–317.
5. D. B. Armstrong, "A Deductive Method for Simulating Faults in Logic Circuits," I.E.E.E. Transactions on Computers, *C-21*, No. 5 (May 1972).
6. H. C. Godoy and R. E. Vogelsberg, "Single Pass Error Effect Determination (Speed)," I.B.M. Technical Disc. Bulletin, *13* (April 1971).
7. S. A. Szygenda, "Problems Associated with the Implementation and Utilization of Digital Simulators and Diagnostic Test Generation Systems," International Symposium on Fault-Tolerant Computing (March 1971).